?????

Unity + PHP + MySQLプロジェクトを構築する前に、いくつか学習しておくべき項目がある。

Awaitable ??

非同期処理について

Unity??Web????????? UnityWebRequest

これまではブラウザでサーバーに接続し、PHPスクリプトを実行してきたが、これからはUnityからリクエストを送信する。そのため、UnityWebRequest クラスを使用して、指定したURLに接続する機能がある。

GET ?????

```
// GETでリクエストを送信
// url: 接続したアドレス
// request: 待機できるオブジェクト
var request = UnityWebRequest.Get(url);
```

使用例:

```
// ユーザーIDでログインしているかどうかを確認
var request = UnityWebRequest.Get("https://api.my-
game.co.jp/api/check_login.php?user_id=45b99aef");
```

ここで:

https://api.my-game.co.jp/api	APIの「根」（Root URL）
check_login.php	サーバーの中で使いたい機能（ログインをチェック）
?user_id=45b99aef	URLパラメータ（ユーザーのID）

もちろん、上記のAPIはあくまでの例であり、ゲームにより変わるので、具体的なAPIはサーバーエンジニアに相談してください。

POST ?????

```
// POSTでリクエストを送信
// url: 接続したアドレス
// data: POST データ (WWWForm)
// request: 待機できるオブジェクト
var request = UnityWebRequest.Post(url, data);
```

使用例:

```
// POSTでデータを送るので、フォームを用意する
// サーバーが期待しているデータに合わせる
var data = new WWWForm();
data.AddField("user_id", "45b99aef");

// ログインAPIのURLへリクエストを送信
var request = UnityWebRequest.Post("https://api.my-game.co.jp/api/check_login.php", data);
```

これ、GET と同じリクエストであるが、POST でデータを送信した例である。

?????

ネットワークの状況やサーバーの混雑により、返事がすぐ来ない場合もあるので、リクエストが終わるまで待機する必要がある。Awaitable を使用し、返事を待つことができる:

```
public async Awaitable CheckLogin(string user)
{
    // POSTでデータを送るので、フォームを用意する
    // サーバーが期待しているデータに合わせる
    var data = new WWWForm();
    data.AddField("user_id", "45b99aef");

    // ログインAPIのURLへリクエストを送信
    var request = UnityWebRequest.Post("https://api.my-game.co.jp/api/check_login.php", data);

    // 返事を待機する
    await request.SendWebRequest();
```

```
    // 返事を処理
}
```

?????

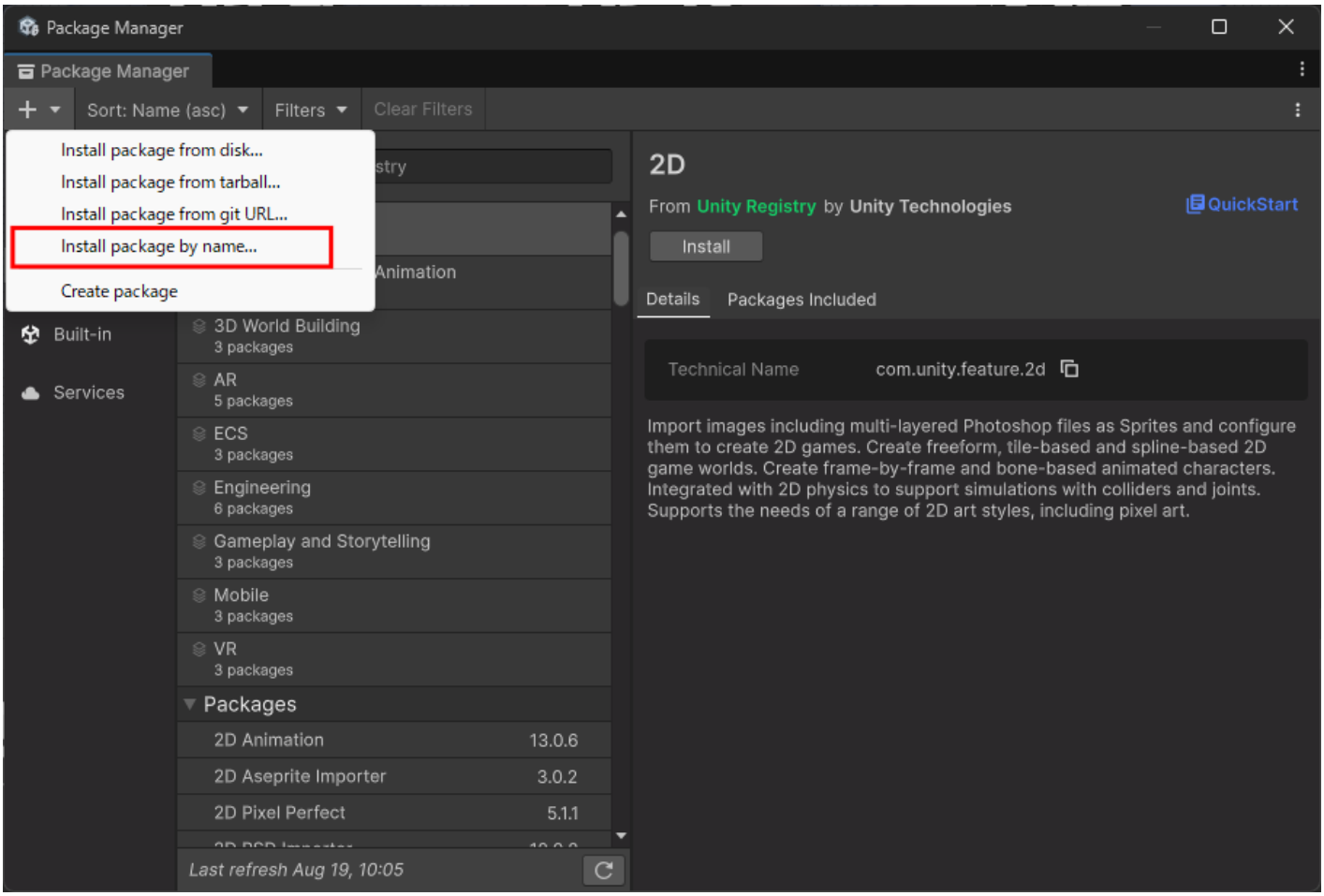
最後、サーバーの返事を確認し、適切に処理する。データが2つの求め方がある：「テキスト（文字列）」、または「バイナリデータ」

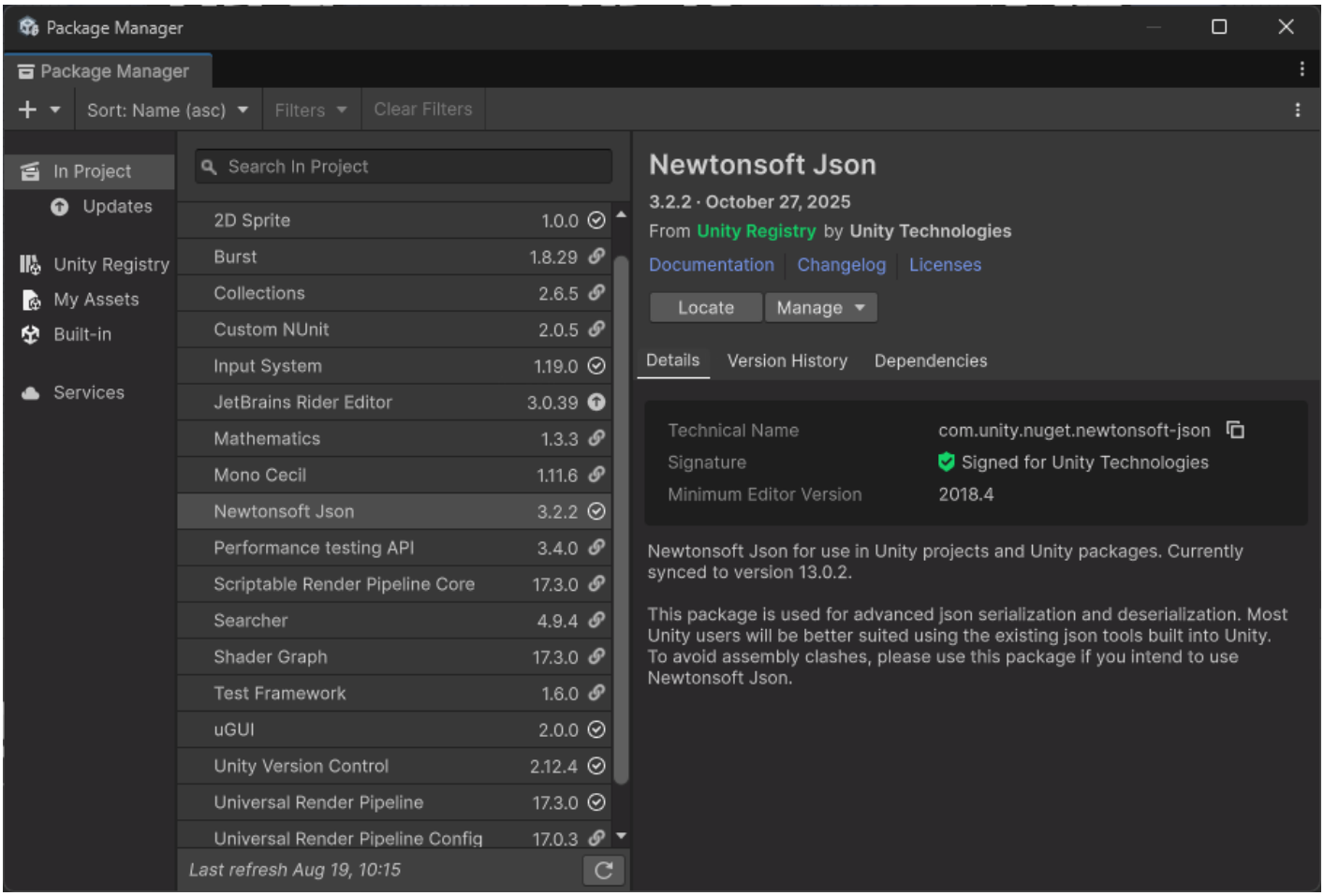
```
public async Awaitable CheckLogin(string user)
{
    // POSTでデータを送るので、フォームを用意する
    // サーバーが期待しているデータに合わせる
    var data = new WWWForm();
    data.AddField("user_id", "45b99aef");

    // ログインAPIのURLへリクエストを送信
    var request = UnityWebRequest.Post("https://api.my-game.co.jp/api/check_login.php", data);

    // 返事を待機する
    await request.SendWebRequest();

    // 返事を処理。以下のどちらか使う
    // 文字列
    string textResponse = request.downloadHandler.text

    // 生のバイナリデータ（画像など）
    bytes[] byteResponse = request.downloadHandler.data;

    // なにかの処理をする
}
```

Unity?JSON???

これから Unity と PHP のやりとりは JSON で行うため、 JSON データを管理しやすくツールを使用しましょう。 Unity は JSON に対応していないが、パッケージとして追加できる。 Package Manager を開き、「 Install Package By Name」を選択し、次のパッケージをインストールしてください:

`com.unity.nuget.newtonsoft-json`


Package Manager
Package Manager
Sort: Name (asc)
Filters
Clear Filters
Install package from disk...
Install package from tarball...
Install package from git URL...
Install package by name...
Create package
Built-in
Services
3D World Building
3 packages
AR
5 packages
ECS
3 packages
Engineering
6 packages
Gameplay and Storytelling
3 packages
Mobile
3 packages
VR
3 packages
Packages
2D Animation 13.0.6
2D Aseprite Importer 3.0.2
2D Pixel Perfect 5.1.1
Last refresh Aug 19, 10:05
2D
From Unity Registry by Unity Technologies
QuickStart
Install
Details
Packages Included
Technical Name com.unity.feature.2d
Import images including multi-layered Photoshop files as Sprites and configure them to create 2D games. Create freeform, tile-based and spline-based 2D game worlds. Create frame-by-frame and bone-based animated characters. Integrated with 2D physics to support simulations with colliders and joints. Supports the needs of a range of 2D art styles, including pixel art.


Package Manager
Package Manager
Sort: Name (asc)
Filters
Clear Filters
In Project
Updates
Unity Registry
My Assets
Built-in
Services
Search In Project
2D Sprite 1.0.0
Burst 1.8.29
Collections 2.6.5
Custom NUnit 2.0.5
Input System 1.19.0
JetBrains Rider Editor 3.0.39
Mathematics 1.3.3
Mono Cecil 1.11.6
Newtonsoft Json 3.2.2
Performance testing API 3.4.0
Scriptable Render Pipeline Core 17.3.0
Searcher 4.9.4
Shader Graph 17.3.0
Test Framework 1.6.0
uGUI 2.0.0
Unity Version Control 2.12.4
Universal Render Pipeline 17.3.0
Universal Render Pipeline Config 17.0.3
Last refresh Aug 19, 10:15
Newtonsoft Json
3.2.2 · October 27, 2025
From Unity Registry by Unity Technologies
Documentation
Changelog
Licenses
Locate
Manage
Details
Version History
Dependencies
Technical Name com.unity.nuget.newtonsoft-json
Signature Signed for Unity Technologies
Minimum Editor Version 2018.4
Newtonsoft Json for use in Unity projects and Unity packages. Currently synced to version 13.0.2.
This package is used for advanced json serialization and deserialization. Most Unity users will be better suited using the existing json tools built into Unity. To avoid assembly clashes, please use this package if you intend to use Newtonsoft Json.

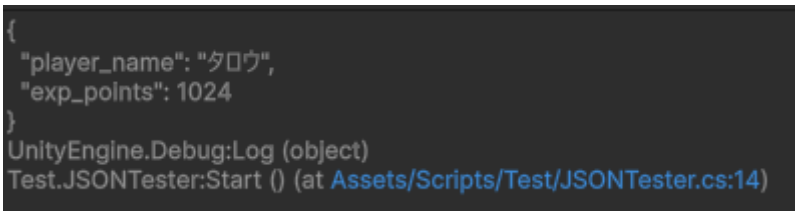
JSON??????

パッケージをインストールすると、 2つの新しいクラスが利用可能になる: `JObject` と `JArray`

`JObject` はディクショナリー（連想配列）として機能する。次のように、キー名を使用して JSON データの各値にアクセスでる。

```
private void Start()
{
    JObject json = new JObject();
    json["player_name"] = "タロウ"; // プレーヤー名
    json["exp_points"]  = 1024;     // 経験値

    Debug.Log(json.ToString());    // 文字列にする
}
```

その結果、コンソールに次のメッセージが表示されます:

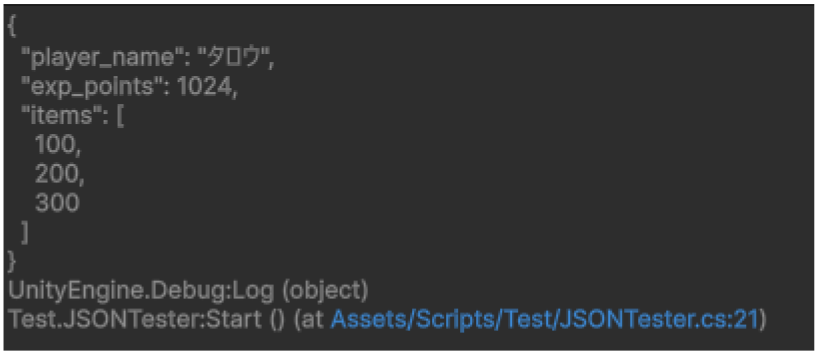


```
{
  "player_name": "タロウ",
  "exp_points": 1024
}
UnityEngine.Debug:Log (object)
Test.JSONTester:Start () (at Assets/Scripts/Test/JSONTester.cs:14)
```

`JArray` は、同じ型の値からなる配列である。リストが必要な場合は、 `JObject` の一部として使用できる。例えば、

```
private void Start()
{
    JObject json = new JObject();
    json["player_name"] = "タロウ"; // プレーヤー名
    json["exp_points"] = 1024;      // 経験値

    JArray items = new JArray();   // 配列
    items.Add(100);
    items.Add(200);
    items.Add(300);

    json["items"] = items;         // JSONに追加

    Debug.Log(json.ToString());    // 文字列にする
}
```

実行すると、以下のようになる:

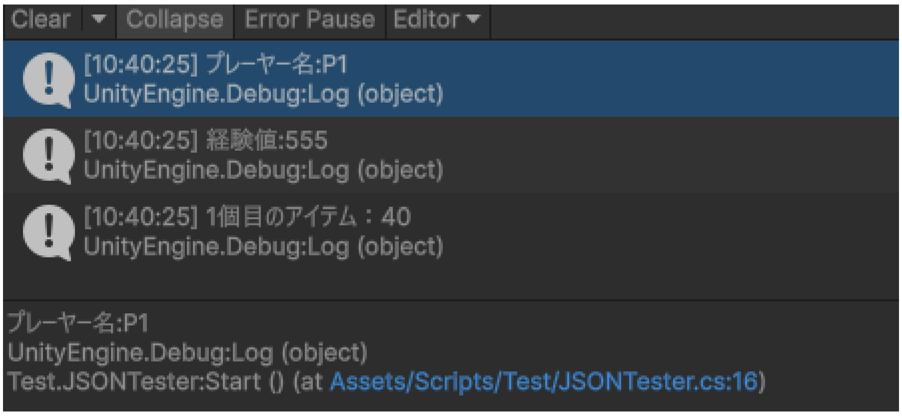


????? JObject ???

前述のように、`JObject.ToString()` を使用すると、JObject を文字列に変換できる。その逆も可能です。`JObject.Parse()` を使用すると、文字列を `JObject` に変換できる。例えば:

```
private void Start()
{
    // JSONの文字列
    string text = "{ \"player_name\" : \"P1\", \"exp_points\" : 555, \"items\" : [40, 30, 20]
}";

    // JObjectへ変換
    JObject json = JObject.Parse(text);

    Debug.Log("プレーヤー名:" + (string)json["player_name"]);
    Debug.Log("経験値:" + (int)json["exp_points"]);

    // JArrayをアクセス
    JArray items = (JArray)json["items"];
    Debug.Log("1個目のアイテム:" + (int)items[0]);
}
```

次のような出力が表示される:



```
[10:40:25] プレーヤー名:P1
UnityEngine.Debug:Log (object)
[10:40:25] 経験値:555
UnityEngine.Debug:Log (object)
[10:40:25] 1個目のアイテム：40
UnityEngine.Debug:Log (object)

プレーヤー名:P1
UnityEngine.Debug:Log (object)
Test.JSONTester:Start () (at Assets/Scripts/Test/JSONTester.cs:16)
```

???????????

本プロジェクトはすでにいくつかの便利な機能を実装されている。好きなメッセージを表示し、ボタンのイベントにも登録できる

???????????????

```
// ただのポップアップを表示する（決定ボタンのみ）
PopupController.Show("ﾀｲﾄﾙ", "詳細ﾒｯｾｰｼﾞ");

// 決定ボタンとキャンセルボタンを表示し、イベントに登録
var popup = PopupController.Show("ﾀｲﾄﾙ", "ﾒｯｾｰｼﾞ", PopupButtons.Ok | PopupButtons.Cancel);
popup.ButtonClicked += OnPopupButton;
```

そして、他のメソッドでイベントを処理する：

```
// ポップアップのボタンを処理
private void OnPopupButton(PopupButtons button)
{
    switch (button)
    {
        // 決定を押した
        case PopupButtons.Ok:
            break;

        // キャンセルを押した
        case PopupButtons.Cancel:
            break;
    }
}
```

Unityで以下のように見えます:

Revision #6
Created 2026-08-04 01:54:45 UTC by Menendez Francisco
Updated 2026-08-19 01:49:06 UTC by Menendez Francisco